

Averaging, Grids and Interpolation

Averaging

- There are many ways...
 - Simple user-defined averages
 - Averaging functions in `MA(.average)`
 - Averaging functions in `cdutil(.averager)`
- This session presents some details on:
 - Averaging over axes
 - Weighted averaging

Simple user-defined averaging

- You can create your own simple averages using arrays, slabs or variables in the usual way:

- Averaging over 4 time steps:

```
>>> t.shape
```

```
(4, 181, 360)
```

```
>>> av=(t[0]+t[1]+t[2]+t[3])/4
```

- Drawbacks:

- Doesn't retain your metadata.
- Cannot average simply across axes within a variable.

MA Averaging

- The MA module has an averaging function:

`MA.average(x, axis=0, weights=None, returned=0)`

- computes the average value of the non-masked elements of x along the selected axis. If $weights$ is given, it must match the size and shape of x , and the value returned is:

$$\frac{\sum (weights_i \cdot x_i)}{\sum weights_i}$$

- elements corresponding to those masked in x or $weights$ are ignored. If `returned`, a 2-tuple consisting of the average and the sum of the weights is returned.

MA Averaging: example

- **To calculate a set of zonal means:**

```
>>> import MA
>>> # data is some array
>>> arr=MA.array(data)
>>> print arr.shape
(1,181,360)
>>> zm=MA.average(arr, axis=2)
>>> print zm.shape
(1,181)
```

The **cdutil** “averager” function

- The “**cdutil.averager()**” function is the key to spatial and temporal averaging in CDAT.
- Masks are dealt with implicitly.
- Powerful area averaging function.
- Provides control over the order of operations (i.e. which dimensions are averaged over first).
- Allows weightings for the different axes:
 - pass your own array of weights for each dimension, use the default (grid) weights or specify equal weighting.

Usage of cdutil.averager

```
result = averager( V, axis=axisoptions,  
                  weights=weightoptions,  
                  action=actionoptions,  
                  returned=returnedoptions,  
                  combinewts=combinewtsoptions)
```

axisoptions has to be a string. You can pass axis='tyx', or '123', or 'x (plev)'.

weightoptions is one of 'generate' | 'weighted' | 'equal' | 'unweighted' | array | Masked Variable

actionoptions is 'average' | 'sum' [Default = 'average'].

You can either return the weighted average or the weighted sum of the data.

***weightoptions:* 'generate' or 'weighted' option**

- Weights are generated using the bounds for the specified axis.
- For latitude and longitude, the weights are calculated using the area (see `grid.getWeights()`).
- For other axes weights are the difference between the bounds (when the bounds are available).
- If the bounds are stored in the file being read in, then those values are used. Otherwise, bounds are generated as long as `cdms.setAutoBounds('on')` is set (the default setting).
- If `cdms.setAutoBounds('off')` is set then an Error is raised.

cdutil.averager – basic usage (1)

```
>>> import cdms, cdutil
>>> f=cdms.open('myvars.xml')
>>> var=f('no10u', time=slice(0,10))
>>> print var
no10u
array( array(10,181,360), type=f, has 651600
      elements)
>>> lon_average=cdutil.averager(var, axis="x")
>>> lon_average.shape
(10, 181)
>>> time_average=av(var, axis="t")
>>> time_average.shape
(181, 360)
```

cdutil.averager – Weights

- Use auto generated weights based on bounds for area averaging:

```
>>> av=cdutil.averager
>>> area_av=av(var, axis="yx", \
               weights=["generate", "generate"])
>> area_av.shape
(10,)
```

```
>>> all_av=av(var, axis="tyx", \
               weights=["generate", "generate", "generate"])
>>> all_av.shape
()
```

- You can use the “area_weights” function to generate a set of weights before averaging.

```
>>> gen_weights = cdutil.area_weights(x)
```


Temporal averaging

- Averaging over time is a special problem in climate data analysis.
- **cdutil** makes the extraction of time averages and climatologies simple.
- Functions for annual, seasonal and monthly averages and climatologies
- User-defined seasons (such as “FMA”).
- Dealt with in detail in the climate-specific tools session.

Grids in CDAT

- Why grids?
- The CDMS RectGrid class
- Regridding (interpolation)
- Horizontal regridding
- Vertical regridding
- SCRIP – regridding Generic grids

Why grids?

- Horizontal grid is very common domain description for a measured/modelled phenomenon.
- A HorizontalGrid represents a latitude-longitude coordinate system. In addition, it optionally describes how lat-lon space is partitioned into cells. Specifically, a HorizontalGrid:
 - consists of a latitude and longitude coordinate axis.
 - may have associated boundary arrays (bounds) describing the grid cell boundaries.
 - may optionally have an associated logical mask.

The CDMS RectGrid class and its operations.

- CDMS includes support for grids, commonly we encounter RectGrids.

- Reading a grid from a variable:

```
>>> grd=var.getGrid()  
>>> lat=grd.getLatitude()
```

- Creating a RectGrid:

```
cdms.createRectGrid(lat, lon, order,  
                    type="generic", mask=None)
```

- For example:

```
>>> lat=cdms.createAxis([1,2,3])  
>>> lat.designateLatitude()  
>>> lon=cdms.createAxis([0,2,4])  
>>> lon.designateLongitude()  
>>> grd=cdms.createRectGrid(lat, lon)
```


More useful grid operations

- Support for Gaussian grids:

```
>>> cdms.createGaussianGrid(nlats, xorigin=0.0,  
    order="yx" )
```

- Creates a Gaussian grid, with shape (nlats, 2*nlats).
- *nlats* is the number of latitudes.
- *xorigin* is the origin of the longitude axis.
- *order* is either “yx” (lat-lon, default) or “xy” (lon-lat)

- Support for diagnostic grids:

```
>>> createZonalGrid(grid)
```

- Creates a zonal grid. The output grid has the same latitude as the input grid, and a single longitude. This may be used to calculate zonal averages via a regridding operation. *grid* is a RectGrid.

```
>>> createGlobalMeanGrid(grid)
```

- Generate a grid for calculating the global mean via a regridding operation. The return grid is a single zone covering the range of the input grid. *grid* is a RectGrid.

Support for other grid types

RectGrid - Associated latitude and longitude are 1-D axes, with strictly monotonic values.

CurveGrid - Latitude and longitude are 2-D coordinate axes (Axis2D).

GenericGrid - Latitude and longitude are 1-D auxiliary coordinate axes (AuxAxis1D)

Curvilinear and Generic Grids

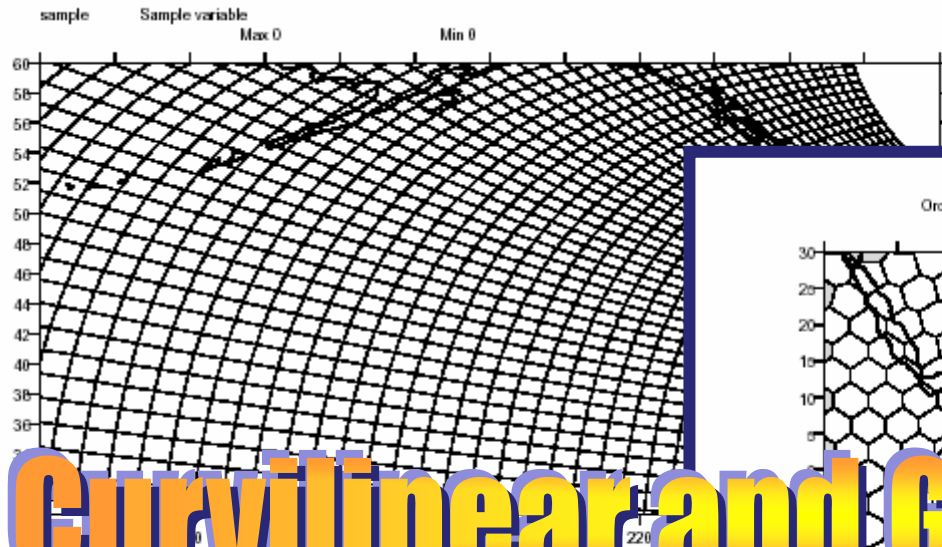


FIGURE 1. Curvilinear grid

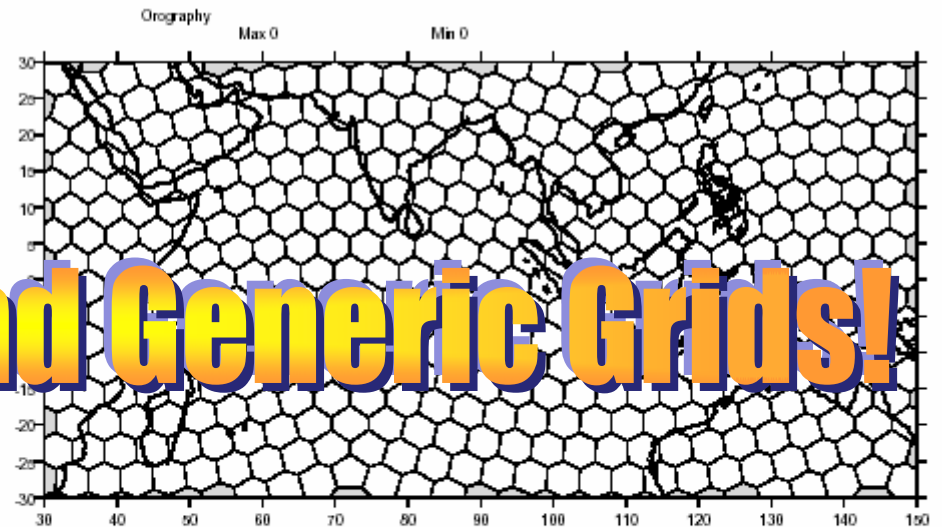


FIGURE 2. Generic grid

Generic grids can be used to represent any of the grid types. The method **toGenericGrid** can be applied to any grid to convert it to a generic representation. Similarly, a rectangular grid can be represented as curvilinear. The method **toCurveGrid** is used to convert a non-generic grid to curvilinear representation:

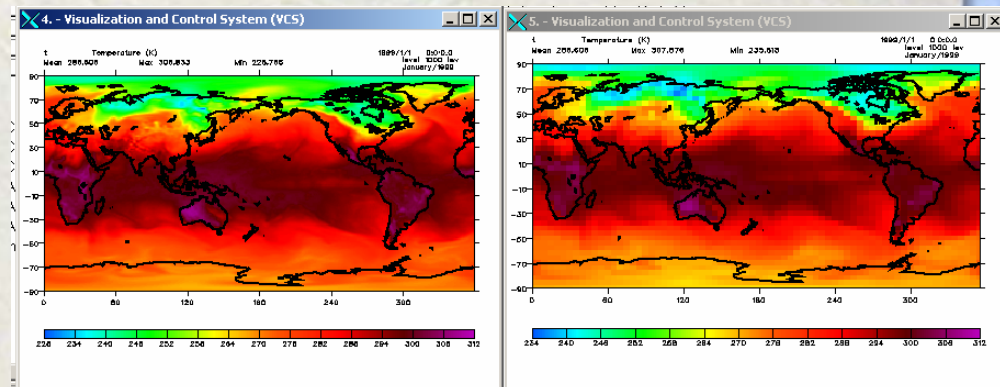
Interpolation (re-gridding)

T regrid variable **u** (from a rectangular grid) to a 96 x 192 rectangular Gaussian grid:

```
>>> import cdms
>>> f=cdms.open( 'mydata.nc' )
>>> var = f( 'temp' )
>>> var.shape
(3, 72, 144)
>>> n48_grid = cdms.createGaussianGrid(96)
>>> var48 = var.regrid(n48_grid)
>>> var48.shape
(3, 96, 192)
```


Using createUniformGrid()

```
#!/usr/local/cdat/bin/python
import cdms
from regrid import Regridder
f = cdms.open('temp.nc')
t= f.variables['t']
ingrid = t.getGrid()
outgrid = cdms.createUniformGrid(-90.0, 46,
                                  4.0, 0.0, 72, 5.0)
regridFunc = Regridder(ingrid, outgrid)
newt = regridFunc(t)
import vcs
vcs.init().plot(t)
vcs.init().plot(newt)
```



Using the Regridder class

- If you are going to use a regridding function repeatedly it is more efficient to create your own regridding function using the Regridder class:

```
>>> import cdms, regrid
>>> f=cdms.open('myfilio.nc')
>>> var1=f('surface_temperature')
>>> var2=f('no2t')
>>> grid1=var1.getGrid()
>>> grid2=var2.getGrid()
>>> regridFunc=regrid.Regridder(grid1, grid2)
>>> regriddedVar1=regridFunc(var1)
>>> diff=regriddedVar1-var2 # gives difference
```


Vertical regridding

- You can regrid pressure-level coordinates in the vertical axis using the **pressureRegrid()** method.
- You need to define, or use an existing, vertical axis.
- Then use the `pressureRegrid` method on the variable you wish to regrid, passing it the new level as the argument:
- If `var` is the variable to regrid and the `newlevs` is the vertical axis to regrid to:

```
>>> var_on_new_levels = var.pressureRegrid(levout)
```

Vertical regridding: an example

```
>>> import MV, cdms
>>> f=cdms.open('temp.ct1')
>>> t=f('t')
>>> t.getLevel()[:]
[ 1000.,  925.,  850.,  775.,  700.,  600.,  500.,
 400.,  300.,  250.,  200.,  150.,  100.,   70.,
 50.,   30.,  20.,  10.,   7.,   5.,   3.,
 2.,   1.,]
>>> t.shape
(1, 23, 181, 360)
>>> nl=cdms.createAxis(MV.array([976.0, 831.0,
221.0]))
>>> nl.designateLevel()
>>> t_regridded=t.pressureRegrid(nl)
>>> t_regridded.shape
(1, 3, 181, 360)
>>> t_regridded.getLevel()[:]
[ 976., 831., 221.,]
```


SCRIP – regridding irregular grids (1)

- CDAT now supports irregular grids that can be interpolated using the **SCRIP** (Spherical Coordinate Re-mapping and Interpolation Package) package (not provided with CDAT) as follows:
 - Obtain or generate the source and target grids in SCRIP NetCDF format. Write a CDMS grid using the **writeScripGrid()** method.
 - Edit the input name list file **scrip_in** to reference the grids and select the method of interpolation, either *conservative*, *bilinear*, *bicubic*, or *distanceweighted*.
 - See the SCRIP documentation for detailed instructions:
<http://climate.lanl.gov/Software/SCRIP/SCRIPusers.pdf>

SCRIP – regridding irregular grids (2)

- At the command line, the SCRIP executable generates a re-mapping file containing the transformation coefficients.
- In CDMS, open the re-mapping file and create a regridding function with the **readRegridding()** method.
- Call the regridding function on the input variable, defined on the source grid.